scientific reports

OPEN

Modeling and verifying clustering properties in a vehicular ad hoc network protocol with Event-B

Patrick Sondi[1]✉, Imed Abbassi[1], Eric Ramat[1], Emna Chebbi[2] & Mohamed Graiet[3]

Vehicular ad hoc network (VANET) routing protocols resort to clustering in order to optimize broadcast traffic flooding. Clustering schemes usually rely on rules which apply to each vehicle in order to reach a targeted organization in a VANET. Most of the literature works which evaluate clustering for VANET focus on performance analysis. However, with autonomous vehicles coming to roadways, more rigorous relationships will be required between clustering rules and the resulting organization, so as to anticipate road safety in a better way. We propose a formal description of the properties which are expected in a VANET, while considering the rules of a given clustering scheme. Using Event-B, we first present a description of the VANET, the vehicles movement and the traffic generated by both routing and application messages. Then, based on an Event-B model of a basic routing protocol of the literature, we describe how the specific rules of a clustering scheme can be modeled along with the properties expected in the resulting organization. Finally, we propose a validation process of the model. This paper aims at showing how our proposals have been applied to the Chain-Branch-Leaf scheme, although they can be adapted to any rule-based clustering scheme for VANET.

Context overview. Thanks to intelligent transport systems (ITS), new information and communication technology applications emerge in the transport sector and its logistics[1]. Vehicular Ad Hoc Network (VANET) extends ITS applications in a context where there is no available telecommunication infrastructure, in order to support road safety and cooperative applications through vehicle-to-vehicle communications[2]. One of the main issues VANET routing protocols must overcome is to achieve self-organization in the network, in a way which not only allows reducing the impact of flooding on the control traffic, but also favors network performance for the applications. For these reasons, clustering is one of the most important techniques used by routing protocols for vehicular ad hoc networks[3]. Particularly, the optimized link state routing (OLSR) protocol and its derivatives use multipoint relaying (MPR) as a clustering technique[4]. Each node selects a minimal subset of its one-hop neighbors which allows it to reach all the nodes located two-hop away. In this way, the multipoint relaying technique leads to a mesh between the vehicles covering the space in which they evolve. Several works show the very good results obtained thanks to MPR techniques for various ad hoc network based applications in open areas[5–7]. However, the MPR technique does not benefit from the particular configuration of road sections which are intrinsically spatially constrained. Chain-Branch-Leaf (CBL), a distributed clustering scheme exploiting this particularity, has been recently proposed[8]. It combines road configuration data, vehicle mobility and link quality indicators in order to build a structure similar to a vehicular network infrastructure, while relying only on vehicle-to-vehicle communications only. Thus, it organizes VANET nodes into a backbone of clusters (which is called "chain"), composed of branch nodes (cluster heads) to which leaf nodes (cluster members) attach themselves. Since 2018, up to thirty recent works referring to CBL have proposed similar mechanisms for various road traffic environments[9–12]. However, the evaluations presented in most of these works, which often rely on analytical or simulation models, focus on the performance of the clustering scheme in terms of ratio between the number of cluster head nodes and the total number of nodes, the number of backbones created in the network, and their impacts on routing overhead, packet delivery ratio, end-to-end delay... etc. Despite the accuracy of such evaluation regarding certain purposes, it is useful to evolve towards evaluation methods which allow providing more guarantees on the properties of a specific protocol or clustering scheme in a VANET environment. Formal methods (e.g., Petri nets[13], finite state machines[14], Event-B[15], etc.) have proven their accuracy in the design of correct systems which properties can be proven using formal tools. They are based on mathematical

[1]Université Littoral Côte d'Opale, LISIC EA 4491, Calais 62228, France. [2]Ecole Centrale de Lille, Cité Scientifique, BP 48, Villeneuve-d'Ascq 59650, France. [3]University of Monastir, B.P 56, Avenue Taher Hadded, Monastir 5000, Tunisia. ✉email: patrick.sondi@univ-littoral.fr

foundations and first order logic to specify and perform reasoning about system properties. In this paper, we use the Event-B formal method to model and verify the properties of the CBL clustering scheme in order to show how such evaluation approach can be applied to VANET protocols.

Related work. Several studies resort to formal modeling for the analysis and evaluation of ad hoc network routing protocols, especially regarding the functionalities which properties can be proven. Some works proposed a complete functional analysis of the protocols, notably DSR (dynamic source routing) protocol in Ref.[16], AODV protocol (Ad hoc On-demand Distance Vector) in Ref.[17] and OLSR (Optimized Link State Routing) protocol in Ref.[18]. These analysis particularly focus on network link detection, route discovery, and relay selection (only for OLSR). However, the spatial relations between the network nodes, and their movement, which are particularly important in VANET, were not considered in these works. Other proposals focus on specific properties of the mechanisms proposed by routing protocols for ad hoc networks. The work in Ref.[19] introduced a series of proposals for the modeling and evaluation of quality of service in communication protocols, using formal methods and languages. Similar proposals were presented for the verification of security properties in ad hoc routing protocols[20,21]. A more recent work[22] also proposes a complete analysis of the Zone Routing Protocol (ZRP), including a detailed description of its formal model and its verification and validation process using Event-B. However, in ad hoc networks the routing protocol is also at the heart of the self-organization of the network. Particularly, in vehicular ad hoc networks, the spatial relations and the movement of the vehicules directly impact that organization. These aspects were not considered in the formal modeling proposed in the previous works. Recently, we presented a preliminary Event-B model for the Chain-Branch-Leaf (CBL) clustering scheme[23] which we proposed for achieving self-organizing in a model of OLSR protocol inspired from[18]. The present work is in keeping with it, but instead of simply extending the previous model[23], it proposes an original model of CBL independently from its implementation in OLSR, and analyses its correctness regarding the specifications of CBL rules. Moreover, it describes how we modeled the spatial relations between the vehicles, their movement, and also network topology discovery and routing functions that are common to most routing protocols for VANET, which could be reused in order to evaluate any other clustering scheme.

The rest of this paper is organized as follows: "Resuts" section describes CBL functioning and a correct-by-construction model for CBL (*CCM4CBL*); the method used during the consistency verification process of the proposed model is described in "Discussion and methods" and "Conclusion and prospective work" section presents our conclusion and prospective for future work.

Results

This section presents the main result of this work, which is a formal model of the Chain-Branch-Leaf clustering scheme allowing verifying its properties in a VANET. As for every clustering schemes, several rules are taken into account by the vehicles involved in the construction of both the clusters and the overall backbone. For example, the uniqueness of the branch node choice for each leaf node is a fundamental property of the Chain-Branch-Leaf organization[24]. However, without a formal description of the CBL scheme, such a requirement can only be envisioned, but not formally expressed nor verified. The assessment of the CBL clustering scheme implies two steps, which are requirement validation for verifying that the protocol specification fulfills the functional needs envisaged by the designer[25], and consistency checking for ensuring that the clustering scheme does not introduce contradictions in the relations linking the nodes. Event-based modeling is particularly suitable for protocol engineering. Event-B is precisely an event-based formal method which has shown its capacity to master the system design complexity through successive refinements[15,26]. The stepwise refinement produces a correct-by-construction model by formally proving the different properties introduced up to each step[26]. Regarding CBL, Event-B provides the tools necessary to perform an incremental verification by checking the properties and constraints defined at each execution step. The different execution steps are characterized by the introduced events. In order to guarantee the invariants preservation by these events, Event-B defines the concept of proof obligation. Therefore, the approach proposed in this paper should allow checking and proving the correctness of the CBL protocol as well as its requirements and properties. Then, once the CBL protocol has been verified, the proposed model guarantees that its execution does not face failure or inconsistency.

Formal description of CBL organization in a VANET. In this section, we first describe CBL functioning along with the related formal definitions. CBL is a distributed algorithm performed by each VANET node[27], considering that the latter may not have a global knowledge of the ad hoc network. Thus, a node can only communicate directly only with its one-hop neighbors which are used as routers in order to reach the rest of the network. A complete description of CBL functioning is presented in Ref.[24]. Figure 1 shows the different elements of the CBL structure in a VANET, which can be summarized as follows:

- *Road configuration* in any road configuration, CBL builds one backbone in each traffic direction. In our example, a 3-lane 2-way highway, CBL builds two separate backbones (Fig. 1).
- *A branch node* is a cluster-head node elected by other nodes (branch or leaf). It is the only one allowed to retransmit the broadcast traffic to the entire network, through its downstream branch, its upstream branch or both.
- *A leaf node* is an ordinary node which attaches itself to the closest branch node. If no branch node is detected, the leaf nodes perform a branch choice process in order to elect one of them.
- *A chain* is a virtual backbone made up of a sequence of branch nodes. Ideally, one chain per traffic direction should be created. On a longitudinal road context such as highways, a chain behaves as a virtual backbone

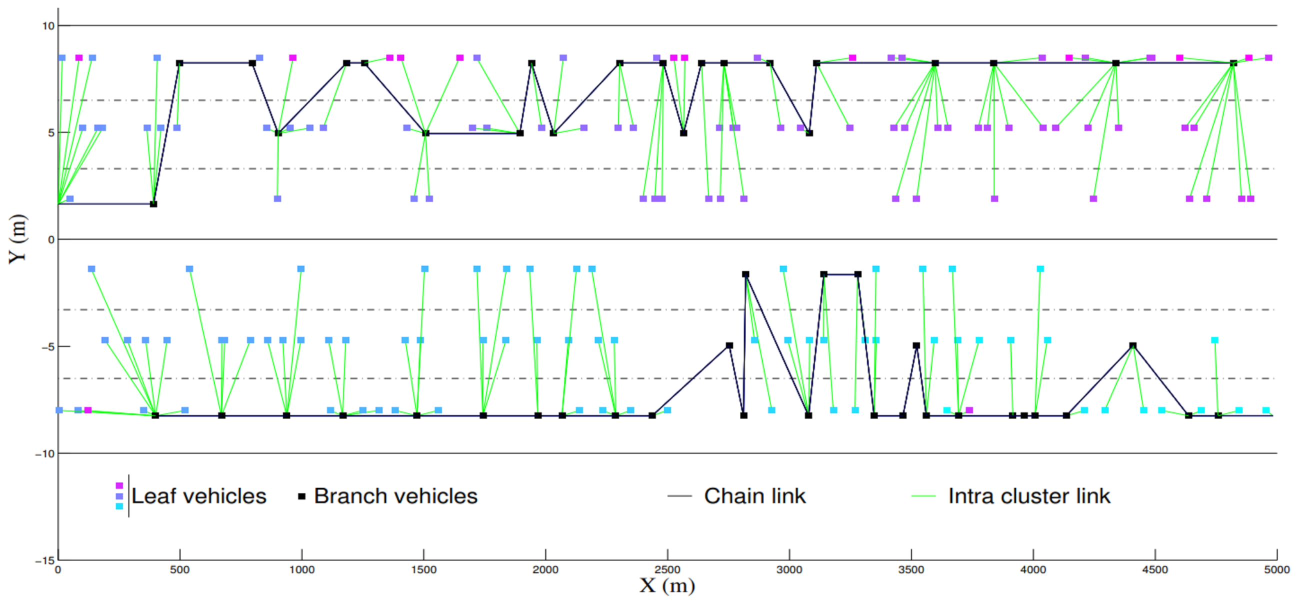


Figure 1. Example of two CBL chains on a 2-way highway.

similar to that which should be obtained with a fixed infrastructure. It provides branch nodes with a path for more than one-hop communications.

In this work, the term *Nodes* refers to the set of all the nodes in the network. Without loss of generality, we will focus in this paper on two types of packets which are sufficient to describe CBL functioning. The hello packets are used by several VANET protocols for neighborhood discovery. The other type of packet refers to the applications traffic. The term *Hello* refers to the set of all the hello packets sent during the VANET scenario. We formally define VANET links using the following functions:

$$links \in \mathbb{P}(Nodes \times Nodes), \tag{1}$$

$$links \cap (Nodes \lhd id) = \varnothing. \tag{2}$$

An element $n1 \mapsto n2$ of the set *links* ($n1 \mapsto n2 \in links$) expresses that the node $n1$ has received a hello packet sent by the node $n2$. We formally define the local neighbors of nodes using the following function:

$$neighbors \in Nodes \rightarrow \mathbb{P}(Nodes). \tag{3}$$

For a given node n, the following property must be fulfilled:

$$\{n\} \times neighbors(n) \subseteq \{n\} \lhd links. \tag{4}$$

One of the main data required by CBL is node position. We assume that each node is aware of its position through a global positioning system such as GPS or Galileo. However, we avoid any terrestrial localization infrastructure since ad hoc networks should not rely on any infrastructure. The node's position is a two-dimensional vector, that the node transmits to its neighbors through hello packets. We formally model the nodes positions data using the following function:

$$positionTable \in Nodes \rightarrow (Nodes \nrightarrow (\mathbb{N} \times \mathbb{N})) \tag{5}$$

Let us consider a node n, the set $positionTable(n)$ includes the position of n and those of its local neighbors. A node can be positioned in the downstream or the upstream side of any of its neighbors. We formally define the downstream/upstream neighbors of nodes using the following functions:

$$down \in Nodes \rightarrow \mathbb{P}(Nodes), \tag{6}$$

$$up \in Nodes \rightarrow \mathbb{P}(Nodes). \tag{7}$$

To determine whether a node is upstream or downstream from another, we use the following operator:

$$DIFF = (\lambda(a \mapsto b) \mapsto (c \mapsto d) \cdot \{a, b, c, d\} \subseteq \mathbb{N} | (a - -c)).$$

Let $n1$ and $n2$ be two neighboring nodes, and let p1 and p2 be their respective positions. We state the following rules

- $n2$ is an upstream node of $n1$ if $DIFF(p1 \mapsto p2)$ is positive.
- $n2$ is a downstream node of $n1$ if $DIFF(p1 \mapsto p2)$ is negative.

When a node does not receive any hello message from a neighbor within a specific period of time, this neighbor is considered to be unavailable and it will be deleted from the neighbors table. This time interval called neighbor expiry time is defined as:

$$Neighbor\_expiry\_time \in \mathbb{N}. \tag{8}$$

In the CBL clustering scheme, a node can be either a leaf or a branch node. We formally define the node types as follows:

$$hasType \in Nodes \rightarrow (Nodes \nrightarrow \{0,1\}) \tag{9}$$

A branch node n ($hasType(n)(n) = 1$) is a cluster-head node elected by the other nodes in its one-hop neighborhood. It emits hello messages like every node. A leaf node n ($hasType(n)(n) = 0$) is an ordinary node which has to connect to the closest branch node. A CBL chain is a sequence of branch nodes. We formally define this chain of branch nodes as follows:

$$chainUP \in Nodes \rightarrow \mathbb{P}(Nodes \times Nodes), \tag{10}$$

$$chainDO \in Nodes \rightarrow \mathbb{P}(Nodes \times Nodes). \tag{11}$$

These two functions are semantically opposed. Hence, *chainUP* (resp. *chainDO*) function defines a local upstream (resp. downstream) chain of branch nodes. If a branch b2 is an upstream node of another branch b1, then b1 is a downstream node of b2. In a CBL organization, each leaf node shall elect its associated branch. We formally define this election as follows:

$$branchChoice \in Nodes \rightarrow \mathbb{P}(Nodes \times Nodes). \tag{12}$$

For a given branch node *n*, branchChoice(n), chainUP(n) (resp. chainDO) refer to the branch choice (if the electing node is a leaf node) or the upstream (resp. downstream) branch nodes (if the electing node is a branch node) included in the neighbors table of *n*. A hello packet can contain different data about its sender, namely the node's position, the node's type, and according to the latter, the elected branch up/down, or the branch choice. To formally define all these data, we use the following functions:

$$helloPosition \in Hello \nrightarrow (\mathbb{N} \times \mathbb{N}) \tag{13}$$

$$helloBrChoice \in Hello \nrightarrow Nodes \tag{14}$$

$$helloChainUP \in Hello \nrightarrow Nodes \tag{15}$$

$$helloChainDO \in Hello \nrightarrow Nodes \tag{16}$$

$$helloSrType \in Hello \nrightarrow \{0,1\} \tag{17}$$

For a given hello packet h sent by a node n, $helloPosition(h)$, $helloBrChoice(h)$, $helloChainUP(h)$, $helloChainDO(h)$ and $helloSrType(h)$ represent respectively the sets of the position, branch choice, elected branch up, elected branch down and the type of n and its neighbors. All these data are very useful in the construction of the CBL scheme. The connection time is the expected communication duration of two nodes according to their movement. More formally, we define the following function:

$$Ctime \in (Nodes \times Nodes) \nrightarrow \mathbb{N} \tag{18}$$

CBL properties and rules. Now that CBL organization has been clarified, it is possible to express the expected properties, and the rules established to that end. The main requirements and resulting properties expected from CBL organization in a VANET are:

- *REQ 1* If a node does not have any local neighbor, it must be a leaf.
- *REQ 2* The branch node choice is made according to leaf nodes only.
- *REQ 3* The self-branch election is not possible.
- *REQ 4* Each branch shall have at most one downstream branch neighbor.
- *REQ 5* Each branch shall have at most one upstream branch neighbor.
- *REQ 6* Each branch shall be elected by at least another node (branch or leaf).
- *REQ 7* A node having a downstream branch shall be of type branch.
- *REQ 8* The electing node of an upstream branch shall be a branch-type one.
- *REQ 9* If a node $n1$ is a downstream branch of a node $n2$, then $n2$ is an upstream branch of $n1$.
- *REQ 10* An upstream branch of a node shall be one of its upstream neighbors.

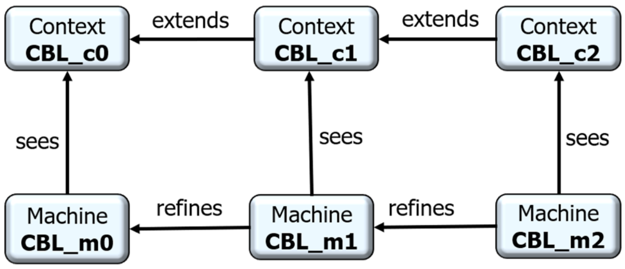


Figure 2. Architecture of the Event-B model for CCM4CBL.

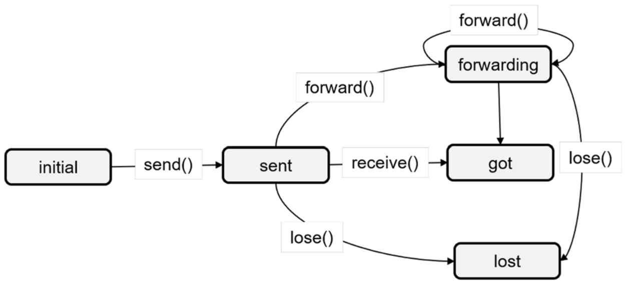


Figure 3. Statechart of packet transfer process.

- *REQ 11* A downstream branch of a node shall be one of its downstream neighbors.
- *REQ 12* One of the neighbors of each leaf node shall become a branch.

Different constraints shall be satisfied while electing branch nodes, such as the following node election rules (NER):

- *NER1* The self-node election is not possible (i.e., the elected node and the electing one must be different).
- *NER2* The elected node must be a neighbor of its electing node.
- *NER3* The elected node must be an upstream neighbor of its electing branch node.
- *NER4* A branch node must have at most one upstream neighbor branch.
- *NER5* When the electing node is a leaf node which does not have any branch node neighbor, it must elect the leaf node neighbor having the maximum connection time as its branch choice.
- *NER6* When the electing node is a branch node which does not have any upstream branch node neighbor, it must elect the upstream neighbor node having the maximum connection time as its upstream branch node.

All nodes are initially leaf nodes. Some of the nodes can be turned into branch nodes, while others must be kept as leaf nodes. In addition, a branch can be turned into a leaf. The type changing rules (TCR) are the following:

- *TCR1* If a leaf node is elected by another node, it must be turned into a branch.
- *TCR2* If the electing node is a branch, then it shall be added to the chain as a downstream branch of the elected one.
- *TCR3* The branch node which overtakes its downstream electing branch shall be turned into a leaf.

A correct-by-construction model of CBL (CCM4CBL). In this section, is presented the proposed Event-B model of the CBL clustering scheme, which implements the aforementioned properties and rules (see "CBL properties and rules" section). Figure 2 illustrates the architecture of the resulting model. It shows three abstraction levels which will be detailed in the following subsections. The two first levels can be used for modeling any other VANET routing protocol.

Level 1: a basic routing protocol model (CBL_c0, CBL_m0). This model inspired from Ref.[18] is an Event-B formal model of a basic routing protocol. It includes the definitions of the set of nodes, that of the links, and also the events related to packet status and operations (Fig. 3). It can be extended and refined in order to model any other routing protocol. Although we rewrote the model (Fig. 4), a similar one is available in the literature[23]. This model has been implemented through the contex *CBL_c0* and the machine *CBL_m0* illustrated in Fig. 2.

```
CONTEXT CBL_0
SETS Packets Nodes STATUS
CONSTANTS default_ttl got lost travelling sent
AXIOMS
axm1: partition(STATUS,{got},{lost},{travelling},{sent})
axm2: default_ttl = 255
...
END
```

```
MACHINE CBL_m0 SEES CBL_c0
VARIABLES
packetStatus packetLocation packetTTL packetHops
packetSource packetDestination
INVARIANTS
inv1:packetStatus ∈ Nodes×Packets⇸STATUS
...
END
```

Figure 4. Level 1: nasic protocol model including a machine and a context.

```
CONTEXT CBL_c1 EXTENDS CBL_c0
CONSTANTS
Hello Neighbor_expiry_time DIFF
AXIOMS
axm1: Hello ⊆ Packets
axm2: Neighbor_expiry_time ∈ ℕ
...
END
```

```
MACHINE CBL_m1 REFINES CBL_m0
SEES CBL_c1
VARIABLES
...
nextNodeToReceiver links neighbors positionTable receptionTime
neighboringTime time lastGotHello helloPosition up down
INVARIANTS
inv1:links ∈ ℙ(Nodes × Nodes) ∧ links ∩ (Nodes ◁ id) = ∅
inv2:nextNodeToReceiver ∈ Packets × Nodes ⇸ Nodes
inv3:neighbors ∈ Nodes → ℙ(Nodes)
inv4:time ∈ Nodes → ℕ
inv5:helloPosition ∈ Hello ⇸ ℕ × ℕ
...
END
```

Figure 5. Level 2: modeling VANET nodes dynamics and communications.

Level 2: modeling VANET node dynamics and communications (CBL_c1, CBL_m1). This second abstraction level of the *CCM4CBL* model formally defines the concepts related to VANET node dynamics and communications, notably neighborhood management, nodes positioning in the road traffic, and packet broadcasting. It consists in the *CBL_m*1 machine and the *CBL_c*1 context (see Fig. 5). In addition to the definitions in "Formal description of CBL organization in a VANET" section, the *CBL_c*1 context extends the initial *CBL_c*0 context by introducing the finite set of all the hello packets ($Hello \subset Packets$). Given the *CBL_c*1 context, the *CBL_m*1 machine refines *CBL_m*0 by introducing the variables, invariants and events modeling VANET node communications, vehicles movement in the road traffic, and neighborhood discovery and management.

Modeling VANET node communications. Communications in a VANET not only include the forwarding of application packets, but also the broadcasting of hello packets for neighborhood discovery and link detection. To formally model ad hoc communications in a VANET, in addition to the definitions proposed in "Formal description of CBL organization in a VANET" section, we introduce the following variables :

- *nextNodeToReceiver* a variable determining the next receiver node of each sent packet (see *inv*2, Fig. 5). This variable is used for forwarding packets from its source to its destination.
- *lastGotHello*: a variable determining the last hello a node received from another. We define this variable as follows:

$$lastGotHello \in Nodes \times Nodes \nrightarrow Hello$$

EVENT broadcast
REFINES sendPacket
ANY source packet TTL
WHERE
grd1:$source \in Nodes \wedge packet \in Packets$
grd2: $TTL \in \mathbb{N}_1 \wedge TTL \leq default\_ttl$
grd3: $packet \in Hello \Rightarrow TTL = 1$
WITH
destinations: $destinations = Nodes \setminus \{source\}$
THEN
...
act6:$packetDestination(packet) := Nodes \setminus \{source\}$
act7:$helloPosition := helloPosition \mathbin{\lhd\!\!-} ((\{packet\} \cap Hello) \times \{positionTable(source)(source)\})$
END

Figure 6. The broadcasting packet event.

- *time* a variable determining the local time of each node (see inv4, Fig. 5)[24].
- *receptionTime* A variable determining the reception time of a packet according to its destination node. We formally define this variable using an invariant as the following function:

$$receptionTime \in Nodes \times Packets \mathrel{+\!\!\!\rightarrow} \mathbb{N}$$

- *neighboringTime* a variable determining the time when a node becomes another's neighbor. We formally define this variable as the following function:

$$neighboringTime \in Nodes \times Nodes \mathrel{+\!\!\!\rightarrow} \mathbb{N}$$

These variables *time*, *receptionTime* and *neighboringTime* are mainly used in order to control the availability of a node's neighbors, as it will be detailed in the neighborhood management phase. The *sendPacket*, *forwardPacket* and *receivePacket* abstract events are also refined. Two refined versions are proposed for the packet sending event, namely sendPacket and broadcast. The first one is used for sending application packets, while the second allows broadcasting of some specific packets such as hello. As Fig. 6 shows, the abstract parameter *destinations* has disappeared from the packet broadcasting event, while new guards and actions have been added. In addition, a witness clause (*with*) is included in order to define the link between the abstract event and the refined one. This witness states that broadcast packets are able to be received by any node in the ad hoc network, with the exception of the sender. This requires a replacement of every occurrences of *destinations* in the action clause with $Nodes \setminus \{source\}$. Action *act*6 illustrates an example of such replacement. Guard *grd*3 ensures that the TTL of a Hello-type packet is equal to 1 during the broadcast process, in order to avoid its forwarding. Action *act*7 expresses that the hello packet sender shall broadcast its current position to all its neighbors. A new parameter T representing the local time at the destination node, and new guards are added in the refined version of the packet receiving event. The guard clause is extended with the following three predicates:

- $(packet \notin Hello \Rightarrow packetLocation(destination \mapsto packet) \mapsto destination \in cls(links))$,
- $T \in \mathbb{N} \wedge time(destination) \leq T$,
- $(packet \in Hello \Rightarrow destination \mapsto packet \mapsto travelling \notin packetStatus)$,

where *cls* refers to the mathematical closure operator. For a relation $R \in A \leftrightarrow A$, $cls(R)$ is the closure of R, which we define as follows: $cls(R) = \bigcup_{i=0}^{\infty} R^i$, where

$$R^i = \begin{cases} A \lhd id, & \text{if } i = 0 \\ (R \,;\, R^{i-1}), & \text{if } i \geq 1. \end{cases}$$

The actions added in the refinement of the reception event, update *time*, *receptionTime* and *lastGotHello* variables. In the refined packet forwarding event (*forwardPacket*), we introduce a new parameter *nextNode* (nextNode $\in$ Nodes), new guards and a new action. The added guards are used to define the connection links between a packet's source/destination and the forwarding node, while the action updates the nextNodeToReceiver variable as follows:

$$nextNodeToReceiver := nextNodeToReceiver \mathbin{\lhd\!\!-} \{packet \mapsto destination \mapsto nextNode\}$$

Modeling the vehicle movement. An *updatePosition* event is defined in the *CBL_m*1 machine in order to model the movement of vehicles (described as members of the set Nodes in our model) on the road. As Fig. 7 shows, this event takes the following parameters as input:

- *node* represents a vehicle whose position shall be updated.
- $XY \in \mathbb{N} \times \mathbb{N}$ refers to the new position of the node.

```
EVENT updatePosition
ANY node XY newUp newDown
WHERE
grd1: $node \in dom(positionTable(node))$
grd2: $XY \in \mathbb{N} \times \mathbb{N}$
grd3: $newUp = \{n, p \cdot n \in neighbors(node) \land n \mapsto p \in positionTable(node) \land DIFF(XY \mapsto p) > 0 | n\}$
...
THEN
act1: $positionTable(node) := positionTable(node) \mathbin{<\!\!+} \{node \mapsto XY\}$
act2: $up(node) := newUp$
act3: $down(node) := newDown$
END
```

Figure 7. Updating vehicles position event.

```
EVENT updateNeighbor
ANY node neighbor T XY H newUp newDown
WHERE
grd1: $node \in Nodes \land neighbor \in Nodes \land node \neq neighbor$
grd2: $node \mapsto neighbor \in dom(lastGotHello)$
grd3: $H = lastGotHello(node \mapsto neighbor) \land H \in dom(helloPosition)$
grd4: $XY = helloPosition(H) \land node \in dom(positionTable(node))$
grd5: $T \in \mathbb{N} \land T \geq time(node)$
grd6: $newUp = (up(node) \setminus \{neighbor\}) \cup \{p \cdot node \mapsto p \in positionTable(node) \land DIFF(p \mapsto XY) > 0 | neighbor\}$
grd7: $newDown = (down(node) \setminus \{neighbor\}) \cup \{p \cdot node \mapsto p \in positionTable(node) \land DIFF(p \mapsto XY) < 0 | neighbor\}$
THEN
act1: $neighbors(node) := neighbors(node) \cup \{neighbor\}$
act2: $links := links \cup \{node \mapsto neighbor\}$
act3: $neighboringTime(node \mapsto neighbor) := T$
act4: $time(node) := T$
act5: $positionTable(node) := (\{neighbor\} \mathbin{\lhd\!\!-} positionTable(node)) \mathbin{<\!\!+} \{neighbor \mapsto XY\}$
act6: $up(node) := newUp$
act7: $down(node) := newDown$
END
```

```
EVENT dropNeighbor
ANY node neighbor
WHERE
grd1: $node \in Nodes$
grd2: $neighbor \in neighbors(node)$
grd3: $node \mapsto neighbor \in dom(neighboringTime)$
grd4: $\{h \cdot h \in Hello \land h \mapsto neighbor \in packetSource \land node \mapsto h \mapsto got \in packetStatus \land node \mapsto h \in dom(receptionTime) \land$
      $receptionTime(node \mapsto h) < neighboringTime(node \mapsto neighbor) + Neighbor\_expiry\_time | h\} = \varnothing$
THEN
act1: $neighbors(node) := neighbors(node) \setminus \{neighbor\}$
act2: $links := links \setminus \{node \mapsto neighbor\}$
act3: $up(node) := up(node) \setminus \{neighbor\}$
act4: $down(node) := down(node) \setminus \{neighbor\}$
act5: $positionTable(node) := \{neighbor\} \mathbin{\lhd\!\!-} positionTable(node)$
END
```

Figure 8. The neighbor updating/removing events.

- *newUp* and *newDown* denote the updated upstream and downstream neighbors of the node.

The last two parameters are checked automatically based on the new node's position using *grd*3 and *grd*4 guards. The event updates the *positionTable* variable by overriding the *positionTable*(*node*) with a set composed of the element $\{node \mapsto XY\}$. The *act*2 and *act*3 actions respectively update the upstream (*newUp*) and downstream (*newDown*) neighbors of the node.

Modeling local neighborhood management. When a node receives a hello packet, it must update its neighbors table according to its content. To model this formally, we add an *updateNeighbor* event in the *CBL_m*1 machine (see Fig. 8). This event has six parameters, three of which (*newUp*, *newDown* and *H*) being automatically computed through the guards, based on the first four parameters. The first two parameters are the source (*neighbor*) and its position *XY* contained in the last hello *H* received from it. Parameters *H* (the last received hello), *newUp* (the updated upstream nodes) and *newDown* (updated downstream nodes) are merely used for simplification. The other two parameters are a destination *node* and its current time *T*. The above parameters are typed through the guards. As stated in Ref.[24], each node periodically broadcasts hello packets to declare its availability to all its neighbors. Thus, when a node does not receive a hello packet from a neighbor within a period of time which equals to the *neighbor_expiry_time*, this neighbor is considered to be unavailable and is deleted from the node's

```
CONTEXT CBL_c2 EXTENDS CBL_c1
CONSTANTS Ctime
AXIOMS
axm1: Ctime ∈ (Nodes × Nodes) ⇸ ℕ . . .
END
```

```
MACHINE CBL_m2 REFINES CBL_m1
SEES CBL_c2
VARIABLES
hasType branchChoice chainUP chainDO
helloBrChoice helloChainUP helloChainDO helloSrType
. . .
END
```

Figure 9. Level 3: modeling CBL properties and rules.

```
EVENT electBranch
ANY electing elected V
WHERE
grd1:electing ∈ Nodes ∧ elected ∈ Nodes ∧ V ∈ {0,1}
grd2:electing ↦ elected ∈ dom(Ctime)
grd3:electing ↦ V ∈ hasType(electing)
// here guards defining NERi rules
THEN
act1:branchChoice(electing) := branchChoice(electing) ⩤ {e.e ∈ {V} ∧ e = 0|electing ↦ elected}
act2:chainUP(electing) := chainUP(electing) ⩤ {e.e ∈ {V} ∧ e = 1|electing ↦ elected}
END
```

Figure 10. Branch election event.

neighbors table. More formally, we create a *dropNeighbor* event in the *CBL_m*1 machine (see Fig. 8). As input parameters, the *dropNeighbor* event has a node and its unavailable neighbor. These parameters are well-defined by *grd*1, *grd*2 and *grd*3 guards. The *grd*3 guard checks the unavailability precondition of the neighboring node before applying the necessary actions.

Level 3: modeling CBL properties and rules (CBL_c2, CBL_m2). The last abstraction level of our CCM4CBL model introduces the specific properties and rules of the CBL clustering scheme in a VANET. As Fig. 9 shows, this implementation level consists in a *CBL_m*2 machine which sees a *CBL_c*2 context.

The *CBL_c*2 context extends the context of the second level (*CBL_c*1) by introducing the concept of node connection time (*Ctime*). The latter is axiomatically defined according to the definition proposed in "Formal description of CBL organization in a VANET" section. Figure 9 also depicts a machine which models CBL properties and rules. This machine sees the *CBL_c*2 context, and refines *CBL_m*1 machine. The presentation of the refinements is organized in four steps, which are: (1) modeling CBL properties, (2) modeling branch nodes election, (3) refining ad hoc communications in a VANET, and (4) modeling CBL chain update.

Modeling CBL properties. New variables are introduced in the *CBL_m*2 machine in order to formally model the specific properties and rules of CBL:

- *hasType* a variable determining the type of the node (Branch or Leaf).
- *branchChoice* a variable determining the branch choice of the node.
- *chainUP* a variable determining the upstream branch node.
- *chainDO* a variable determining the downstream branch node.

The type of each variable is defined using a typing invariant according to the definitions introduced in "Formal description of CBL organization in a VANET" section. Given the variables, constants and sets introduced, CBL properties (*req*1, *req*2, ···, *req*12) can be formally expressed as invariants in Event-B. These invariants are illustrated in Table 1.

Modeling branch nodes election. This election is a key step in the construction of the CBL structure. As stated in Ref.[24], two election types are possible: branch choice by leaf nodes, and upstream branch election by branch nodes. To model these two operations formally, we created an *electBranch* event, taking three parameters as input (see Fig. 10). The first two parameters, *electing* and *elected*, refer to both the electing and elected nodes. The other parameter, *V*, is a binary number ($V \in \{0, 1\}$) used to simplify the verification of the *NERi* rules while electing branch nodes. To achieve this verification, each *NERi* rule is defined as a guard in the *electBranch* event. Each

Property	Corresponding Event-B invariant
REQ 1	$\forall n \cdot n \in Nodes \wedge neighbors(n) = \varnothing \Rightarrow n \mapsto 0 \in hasType(n)$
REQ 2	$\forall n, a \cdot n \in Nodes \wedge a \in dom(branchChoice(n)) \Rightarrow a \mapsto 0 \in hasType(n)$
REQ 3	$\forall n \cdot n \in Nodes \Rightarrow (chainUP(n) \cup branchChoice(n)) \cap (Nodes \lhd id) = \varnothing$
REQ 4	$\forall n \cdot n \in Nodes \Rightarrow chainUP(n) \in Nodes \rightarrowtail\!\!\!\!\!\!\;+ Nodes$
REQ 5	$\forall n \cdot n \in Nodes \Rightarrow chainDO(n) \in Nodes \rightarrowtail\!\!\!\!\!\!\;+ Nodes$
REQ 6	$\forall n \cdot n \in Nodes \wedge n \mapsto 1 \in hasType(n) \Rightarrow (chainUP(n) \cup branchChoice(n)) \rhd \{n\} \neq \varnothing$
REQ 7	$\forall a, n \cdot a \in Nodes \wedge a \mapsto n \in chainDO(n) \Rightarrow a \mapsto 1 \in hasType(n)$
REQ 8	$\forall a, n \cdot a \in Nodes \wedge a \mapsto n \in chainUP(n) \Rightarrow a \mapsto 1 \in hasType(n)$
REQ 9	$\forall a, n \cdot a \in Nodes \wedge a \mapsto n \in chainUP(n) \Rightarrow a \in down(n)$
REQ 10	$\forall a, n \cdot a \in Nodes \wedge a \mapsto n \in chainDO(n) \wedge n \mapsto a \in chainUP(n) \Rightarrow a \in up(n)$
REQ 11	$\forall a, n \cdot a \in Nodes \wedge a \mapsto n \in chainDO(n) \wedge a \in up(n) \Rightarrow n \mapsto a \in chainUP(n)$
REQ 12	$\forall n, b \cdot n \in Nodes \wedge n \mapsto b \in branchChoice(n) \Rightarrow b \in neighbors(n)$

Table 1. Formal modeling of CBL properties with Event-B as invariants.

one of these guards is detailed below. The first rule, *NER*1, states that a node (leaf or branch) cannot elect itself as a branch, while the second one (*NER*2) stipulates that the elected node must be a neighbor of its electing one. These two rules are defined as follows :

$$elected \neq electing \wedge elected \in neighbors(electing) \wedge \ electing \mapsto 0 \in hasType(electing).$$

Rule *NER*3 states that the elected node must be an upstream neighbor of its electing node, which shall be a branch. This rule is expressed through the following predicate:

$$electing \mapsto 1 \in hasType(electing) \ \Rightarrow elected \in up(electing).$$

Rule *NER*4 is used to ensure the uniqueness of the CBL chain, in order to avoid parallel chains in the same area. Hence, it expresses that a branch node must have at most one upstream neighbor branch. We define this rule as follows:

$$V = 1 \Rightarrow chainUP(electing) \rhd up(electing) = \varnothing.$$

Rule *NER*5 states that, when there is no neighboring branch node, the electing leaf must choose the neighbor having the maximum connection time to itself. We formally define this constraint through the following predicate:

$$\begin{aligned} & elected \mapsto 0 \in hasType(electing) \wedge V = 0 \\ & \quad \Rightarrow \neg(\exists n \cdot n \in neighbors(electing) \wedge electing \mapsto n \in dom(Ctime) \wedge Ctime(electing \mapsto elected) < Ctime(electing \mapsto n)). \end{aligned}$$

Rule *NER*6 concerns the connection time of the elected upstream branch. This rule is applied when the electing branch node does not have any upstream branch-type neighbor. It is formally defined similarly to rule *NER*5 as follows:

$$\begin{aligned} & \{elected \mapsto 0\} \subseteq hasType(electing) \wedge V = 1 \\ & \quad \Rightarrow \neg(\exists n \cdot n \in up(electing) \wedge electing \mapsto n \in dom(Ctime) \wedge Ctime(electing \mapsto elected) < Ctime(electing \mapsto n)). \end{aligned}$$

Refining VANET node communications. As stated previously, CBL is a distributed algorithm which builds a backbone of branch nodes, and clusters of leaf nodes around the latter. In order to achieve this task, CBL relies only on the information exchanged through hello packets. A hello packet contains information about its sender, such as its position, its type (branch or leaf), its branch choice when it is a leaf node, its upstream and downstream branch nodes when it is a branch node, and its direct neighbors. The following variables are added:

- *helloBrChoice* a variable determining the branch choice of the hello sender.
- *helloChainUP* a variable determining the upstream branch of the hello sender.
- *helloChainDO* a variable determining the downstream branch of the hello sender.
- *helloSrType* a variable determining the type of the hello sender (branch or leaf).

These variables are defined as partial functions using invariants in the *CBL_m*2 machine according to their definition (see "Formal description of CBL organization in a VANET" section). The packet broadcasting event is refined in order to integrate the information contained in the hello packet. A guard related to CBL property *REQ*12 is added in order to enforce the leaf nodes which have neighbors, so that they can choose their branch nodes before broadcasting their hello packet:

$$neighbors(source) \neq \varnothing \wedge source \mapsto 0 \in hasType(source) \Rightarrow 1 \in hasType(source)[neighbors(source)].$$

New substitution actions are also introduced in the refined version of the *broadcast* event, in order to update the status of the *helloBrChoice*, *helloChainUP*, *helloChainDO* and *helloSrType* variables as follows:

$$\begin{aligned}
&helloBrChoice := helloBrChoice \Leftarrow ((\{packet\} \cap Hello) \times ran(\{source\} \lhd branchChoice(source)))\\
&helloChainUP := helloChainUP \Leftarrow ((\{packet\} \cap Hello) \times ran(\{source\} \lhd chainUP(source)))\\
&helloSrType := helloSrType \Leftarrow ((\{packet\} \cap Hello) \times \{hasType(source)(source)\})\\
&helloChainDO := helloChainDO \Leftarrow ((\{packet\} \cap Hello) \times ran(\{source\} \lhd chainDO(source)))
\end{aligned}$$

Modeling CBL chain update. Several events trigger the update of the CBL chain locally at the VANET node level, such as the unavailability of some neighbors, or the overtaking of a branch node by its upstream branch node. In this work, we consider three events, namely: a neighbor which is no longer available, changes in the nodes' positions, and a received hello packet indicating some changes in the CBL chain (election of new branch up/down, branch choice, etc.). In order to model these events formally, we refine *updatePosition*, *dropNeighbor* and *updateNeighbor* abstract events, and introduce two new events, *turnIntoBranch* and *turnIntoLeaf*, which are used to change the node type according to TCR rules ("CBL properties and rules" section). The *updateNeighbor* variable is refined as follows:

$$\begin{aligned}
hasType(node) &:= newTypes\\
chainDO(node) &:= newChainDO\\
chainUP(node) &:= newChainUP'\\
branchChoice(node) &:= newBrChoice
\end{aligned}$$

where *newTypes*, *newChainDO*, *newChainUP* and *newBrChoice* refer to the parameters introduced in the event's refinement, and are the new values of functions *hasType*(*node*), *chainDO*(*node*), *chainUP*(*node*) and *branchChoice*(*node*) after updating or adding the information (type, branch choice, and so on) about the neighbor. For example, the values of *newChainUP* and *newChainDO* parameters are computed as follows:

$$\begin{aligned}
&newChainUP = chainUP(node) \Leftarrow (\{neighbor\} \times ran(\{H\} \lhd helloChainUP))\\
&newChainDO = \{x \mapsto y | y \mapsto x \in newChainUP\}
\end{aligned}$$

The other parameters are computed similarly. The *dropNeighbor* refined event allows the local update of the CBL chain in case of unavailability of a neighbor. Compared to its abstract version, it introduces the following actions:

$$\begin{aligned}
&branchChoice(node) := branchChoice(node) \setminus \{node \mapsto neighbor\}\\
&chainUP(node) := chainUP(node) \setminus \{node \mapsto neighbor\}\\
&chainDO(node) := chainDO(node) \setminus \{node \mapsto neighbor\}\\
&hasType(node) := \{neighbor\} \ntriangleleft hasType(node)
\end{aligned}$$

Updating the position of a node *Ni* requires checking whether *Ni* has overtaken the node *Nj* that elected it as its upstream branch node. In this case, the chain link between *Ni* and *Nj* must be deleted by adding an action in the *updatePosition* event:

$$chainUP(node) := newChainUP$$

where *newChainUP* is a new parameter referring to the updated upstream branch node, automatically computed as follows:

$$(down(node) \setminus newDown) \ntriangleleft chainUP(node) \tag{19}$$

Provided that the following conditions (expressed as a guard) are satisfied:

$$\begin{aligned}
&(\forall a \cdot a \mapsto node \in chainDO(node) \wedge node \mapsto a \in chainUP(node) \Rightarrow a \in newUp) \wedge\\
&(\forall a \cdot a \mapsto node \in chainDO(node) \wedge a \in newUp \Rightarrow node \mapsto a \in newChainUP) \wedge (newChainUP \in Nodes \rightarrowtail\!\!\!\!\!\!\rightarrow Nodes) \wedge\\
&(\forall n \cdot n \in Nodes \wedge n \mapsto 1 \in hasType(n) \Rightarrow (newChainUP \cup branchChoice(node)) \rhd \{n\} \neq \varnothing)
\end{aligned}$$

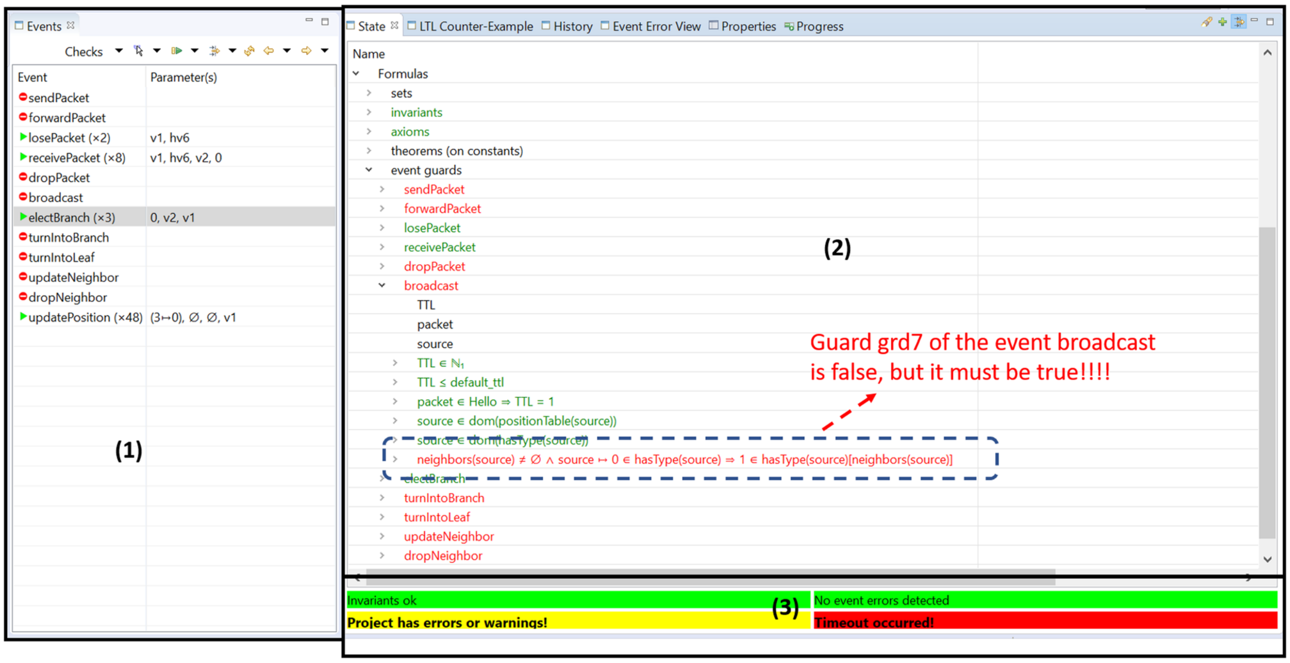
Event *turnIntoBranch* allows turning a leaf node into a branch node when it has been elected by at least one of its neighbors (see grd1, Fig. 11). This type changing occurs after receiving a hello packet from a neighbor. Guard *grd*3 checks CBL property *req*6 which states that self-election is not authorized (a node cannot be elected by itself). Event *turnIntoLeaf* allows turning a branch node into a leaf when the latter overtakes its electing downstream branch node after updating the node's position (see Fig. 11). Guard *grd*1 defines a precondition of this type changing according to rule *TCR*3.

Discussion and methods

The methods adopted during the validation of the *CCM4CBL* model rely on a two-step verification approach: validation of the model by animating it using the `ProB` model-checker, and proving its correctness by discharging proof obligations.

EVENT turnIntoBranch **ANY** leaf electing
WHERE
grd1:$(branchChoice(leaf) \cup chainUP(leaf))$
grd2:$leaf \mapsto 0 \in hasType(leaf)$
grd3:$neighbors(leaf) \neq \varnothing$
grd4:$leaf \notin dom(branchChoice(leaf))$
grd5:$electing \mapsto leaf \in chainUP(leaf) \Rightarrow chainDO(leaf) \rhd down(leaf) = \varnothing$
THEN
act1:$hasType(leaf) := hasType(leaf) \lhd\!\!- \{leaf \mapsto 1\}$
act2:$chainDO(leaf) := chainDO(leaf) \lhd\!\!- \{x \mapsto y | x \mapsto y \in \{electing \mapsto leaf \wedge y \mapsto x \in chainUP(leaf)\}$
END
EVENT turnIntoLeaf **ANY** node
WHERE
grd1:$\{node\} \lhd chainDO(node) \neq \varnothing \wedge chainUP(node) \rhd \{node\} = \varnothing$
grd2:$node \mapsto 1 \in hasType(node)$
grd3:$finite(ran(\{node\} \lhd chainDO(node)) \cap down(node))$
grd4:$card(ran(\{node\} \lhd chainDO(node)) \cap down(node)) \leq 1$
THEN
act1:$hasType(node) := hasType(node) \lhd\!\!- \{node \mapsto 0\}$
act2:$chainDO(node) := \{node\} \lhd chainDO(node)$
END

Figure 11. Node type update events.



Event	Parameter(s)
⊖sendPacket	
⊖forwardPacket	
▶losePacket (×2)	v1, hv6
▶receivePacket (×8)	v1, hv6, v2, 0
⊖dropPacket	
⊖broadcast	
▶electBranch (×3)	0, v2, v1
⊖turnIntoBranch	
⊖turnIntoLeaf	
⊖updateNeighbor	
⊖dropNeighbor	
▶updatePosition (×48)	(3↦0), ∅, ∅, v1

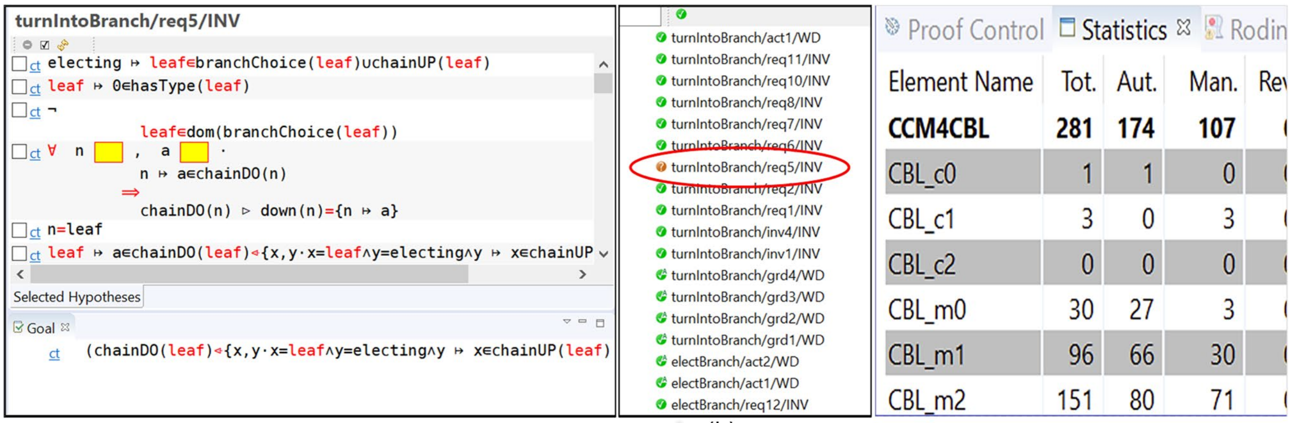
```
Name
Formulas
  sets
  invariants
  axioms
  theorems (on constants)
  event guards
    sendPacket
    forwardPacket
    losePacket
    receivePacket
    dropPacket
    broadcast
      TTL
      packet
      source
      TTL ∈ ℕ₁
      TTL ≤ default_ttl
      packet ∈ Hello ⇒ TTL = 1
      source ∈ dom(positionTable(source))
      source ∈ dom(hasType(source))
      neighbors(source) ≠ ∅ ∧ source ↦ 0 ∈ hasType(source) ⇒ 1 ∈ hasType(source)[neighbors(source)]
    electBranch
    turnIntoBranch
    turnIntoLeaf
    updateNeighbor
    dropNeighbor
```

Guard grd7 of the event broadcast is false, but it must be true!!!!

Invariants ok
No event errors detected
Project has errors or warnings!
Timeout occurred!

Figure 12. ProB animation window of the *CCM4CBL* model.

Animation-based validation. The `ProB`[28] animation tool allows both the validation of the requirements, and the detection of errors in order to fix them, before starting the proof phase which can be long and complex. It cannot be performed on an abstract Event-B specification, and requires a concrete model. For that reason, we created a new *CBL_m3* machine by extending the *CBL_m2* machine. This machine does not introduce new events or variables. It sees a concrete *CBL_c3* context, which is an extension of the *CBL_c2* context. New constants and axioms are defined in this extended context for the concretization of all the sets and functions introduced in the contexts of the proposed *CCM4CBL* model. Figure 12 depicts the `ProB` animation window which is composed of three main parts. The first part (1) describes event triggering and constraint checking. The second part (2) presents the status of the model. The last part (3) allows signaling potential invariant violations and specification errors. After setting up the animation context, only the initialization event is enabled. After that, *updatePosition* and *broadcast* events are successfully activated. Once a hello packet is broadcast by a node, the *receivePacket* and *losePacket* events are enabled. Event *updateNeighbor* is triggered several times after executing *receivePacket*. Regarding the animation scenario Fig. 12 shows, we note that *broadcast* and *turnIntoBranch* events are disabled after executing both events *updateNeighbor* and *electBranch*. No node will be turned into a branch and the clustering scheme cannot proceed. This situation suggests that a given node is not able to broadcast hello



turnIntoBranch/req5/INV

```
ct electing ↦ leaf∈branchChoice(leaf)∪chainUP(leaf)
ct leaf ↦ 0∈hasType(leaf)
ct ¬
        leaf∈dom(branchChoice(leaf))
ct ∀ n ▢ , a ▢ ·
        n ↦ a∈chainDO(n)
       ⇒
        chainDO(n) ▷ down(n)={n ↦ a}
ct n=leaf
ct leaf ↦ a∈chainDO(leaf)⩤{x,y·x=leaf∧y=electing∧y ↦ x∈chainUP
```

Selected Hypotheses

Goal

```
ct  (chainDO(leaf)⩤{x,y·x=leaf∧y=electing∧y ↦ x∈chainUP(leaf)
```

turnIntoBranch/act1/WD
turnIntoBranch/req11/INV
turnIntoBranch/req10/INV
turnIntoBranch/req8/INV
turnIntoBranch/req7/INV
turnIntoBranch/req6/INV
turnIntoBranch/req5/INV
turnIntoBranch/req2/INV
turnIntoBranch/req1/INV
turnIntoBranch/inv4/INV
turnIntoBranch/inv1/INV
turnIntoBranch/grd4/WD
turnIntoBranch/grd3/WD
turnIntoBranch/grd2/WD
turnIntoBranch/grd1/WD
electBranch/act2/WD
electBranch/act1/WD
electBranch/req12/INV

Element Name	Tot.	Aut.	Man.	Re
CCM4CBL	**281**	**174**	**107**	
CBL_c0	1	1	0	
CBL_c1	3	0	3	
CBL_c2	0	0	0	
CBL_m0	30	27	3	
CBL_m1	96	66	30	
CBL_m2	151	80	71	

(a) (b) (C)

Figure 13. Examples of proof obligations and proof statistics in Rodin.

packets to inform the neighbors of its branch choice. It is caused by guard *grd*7 (see Fig. 12), which states that one of the neighbors of each leaf shall be a branch. In order to solve this issue, we modified the related guard:

$$neighbors(source) \neq \varnothing \wedge source \mapsto 0 \in hasType(source) \Rightarrow \{source\} \lhd branchChoice(source) \neq \varnothing.$$

We used the ProB counter-examples as guides to rectify our model and trace back the specification errors, which could have caused the prover failures. The performed rectifications concern invariant violations and both guard and action alterations. We validated our model based on representative scenarios, including several cases which had not been treated previously.

Correctness of the model by discharging proof obligations. In order to verify the model during its design process, proof obligations (POs) are discharged in a way guaranteeing that:

- Model initialization leads to a state where the invariant is valid.
- When the machine is in a state where the invariant is valid, every enabled event leads to a state preserving this validity.
- The concrete events can only occur in the circumstances in which the abstract events occur.
- The occurrence of any concrete event implies an occurrence of the related abstract event in such a way that the state verifies all related invariants.

POs refer to the proofs applicable to an Event-B model. Figure 13 illustrates examples of proof obligations. Those discharged are marked with ✓A, while those undischarged can be recognized thanks to the symbol ?A. The POs which are discharged automatically also show an "A" letter. Some proof obligations need interactive discharging by users, when the automatic prover cannot perform it (Fig. 13c shows the repartition between automatically and manually discharged POs in our model). For example, using the cardinal operator implies a finite set as operand, thus making more difficult the discharging of proof obligations. As well, the universal (∀) and existential (∃) quantifiers may be sources of concerns for the instantiation of the quantified hypothesis. In the proposed model, an example of such a PO is *turnIntoBranch*/*req*5/*INV*, which ensures that the *turnIntoBranch* event preserves the *req*5 invariant of the *CBL_m*2 machine. As depicted in Fig. 13a, the sequent of such a PO is unproved due to a lack of hypothesis. For that reason, we modified the *turnIntoBranch* event, shown in Fig. 11, by adding the following new guard, which let to success:

$$electing \mapsto leaf \in chainUP(leaf) \Rightarrow chainDO(leaf) \rhd down(leaf) = \varnothing.$$

Conclusion and prospective work

In this paper, we have proposed an approach using Event-B to validate the properties and rules of a VANET routing protocol. Targeting the CBL clustering scheme, a correct-by-construction model *CCM4CBL* is proposed for that purpose as a proof of concept. This model includes three abstraction levels. The first one is an initial specification containing the basic functions of any network routing protocol. The second level introduces specific concepts of the VANET environment such as spatial relations between the vehicles according to their positions, vehicle movement, and management of both routing tables and vehicle communications. At this step, the proposed model can be reused for modeling any VANET routing protocol. The last level formally defines the specific properties and rules of the CBL clustering scheme regarding VANET organization. The proposed model is gradually verified using proof obligation mechanisms offered by the event-B method and finally validated using the ProB animator to repair several behavioral errors. These processes are illustrated thanks to concrete examples. In our future work, we will target the coupling of this formal approach to others, such as discrete event modeling and simulation, in order to enhance VANET protocols verification and evaluation, particularly regarding scalability and security properties.

Code availabilty

The code source is available.

Received: 12 June 2021; Accepted: 19 August 2021
Published online: 02 September 2021

References

1. Sussman, J. S. *Perspectives on Intelligent Transportation Systems (ITS)* (Springer, 2008).
2. Ghahramani, S. A. A. G., Hemmatyar, A. M. A. & Kavousi, K. A network model for vehicular ad hoc networks: An introduction to obligatory attachment rule. *IEEE Trans. Netw. Sci. Eng.* **3**, 82–94 (2016).
3. Senouci, O., Harous, S. & Aliouat, Z. Survey on vehicular ad hoc networks clustering algorithms: Overview, taxonomy, challenges, and open research issues. *Int. J. Commun. Syst.* https://doi.org/10.1002/dac.4402 (2020).
4. Qayyum, A., Viennot, L. & Laouiti, A. Multipoint relaying for flooding broadcast messages in mobile wireless networks. In *Proc. 35th annual Hawaii International Conference on System Sciences*, 3866–3875 (IEEE, 2002).
5. Lebedev, D. *Ad Hoc Networks: Study of Protocol Behaviour*. Ph.D. thesis, X - Polytechnique (2006).
6. Maccari, L., Maischberger, M. & Cigno, R. L. Where have all the mprs gone? on the optimal selection of multi-point relays. *Ad Hoc Netw.* **77**, 69–83. https://doi.org/10.1016/j.adhoc.2018.04.012 (2018).
7. Sondi, P., Gantsou, D. & Lecomte, S. Performance evaluation of multimedia applications over an OLSR-based mobile ad hoc network using OPNET. In *12th International Conference on Computer Modelling and Simulation (UKSim)*, 567–572 (IEEE, 2010). https://doi.org/10.1109/uksim.2010.109.
8. Rivoirard, L., Wahl, M., Sondi, P., Berbineau, M. & Gruyer, D. CBL: A clustering scheme for VANETs. In *VEHICULAR 2017—The Sixth International Conference on Advances in Vehicular Systems, Technologies and Applications*, 19 (2017). https://hal.archives-ouvertes.fr/hal-01573625.
9. XiangJi, G. F. H. S. L. C. & Yu, H. Efficient and reliable cluster-based data transmission for vehicular ad hoc networks. *Mobile Inf. Syst.* **2018**, 9826782 (2018).
10. Sasirekha, S. P., Amudhavalli, P. & Sherubha, P. Efficient route map based ant colony optimization for route discovery in vanet. In *2019 International Conference on Communication and Signal Processing (ICCSP)*, 0455–0459 (2019).
11. Ahmad, M. *et al.* Optimized clustering in vehicular ad hoc networks based on honey bee and genetic algorithm for internet of things. *Peer-to-Peer Netw. Appl.* **13**, 532–547 (2020).
12. Fan, X., Liu, D., Fu, B. & Wen, S. Optimal relay selection for uav-assisted v2v communications. *Wirel. Netw.* https://doi.org/10.1007/s11276-021-02644-9 (2021).
13. Girault, C. & Valk, R. *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications* (Springer, 2013).
14. Chow, T. S. Testing software design modeled by finite-state machines. In *IEEE Transactions on Software Engineering*, 178–187 (1978).
15. Abrial, J. *Modeling in Event-B: System and Software Engineering* (Cambridge University Press, 2010).
16. Méry, D. & Singh, N. *Stabilization, Safety, and Security of Distributed Systems* 401–415 (Springer, 2011).
17. Singh, A. & Singh, V. Formal modeling of distance vector routing protocol using Event-B. *Electron. Electr. Eng.* **3**, 91–98 (2013).
18. Kamali, M. & Petre, L. Modelling link state routing in Event-B. In *2016 21st International Conference on Engineering of Complex Computer Systems (ICECCS)*, 207–210 (2016).
19. Cavalli, A. Inénierie des protocoles et qualité de service. *Collection IC2 Réseaux et Télécoms* (2001).
20. Lee, G. How to formally model features of network security protocols. *Int. J. Security Appl.* **8**, 423–432 (2014).
21. Diaz, J., Arroyo, D. & Rodriguez, F. B. A formal methodology for integral security design and verification of network protocols. *J. Syst. Softw.* **89**, 87–98. https://doi.org/10.1016/j.jss.2013.09.020 (2014).
22. Fu, C. & Zheng, K. Formal modeling and analysis of ad hoc zone routing protocol in Event-b. *Int. J. Softw. Tools Technol. Transfer* **21**, 165–181. https://doi.org/10.1007/s10009-017-0463-4 (2019).
23. Chebbi, E., Sondi, P. & Ramat, E. A formal model for the chain-branch-leaf clustering scheme in OLSR based vehicular ad hoc networks using Event-B. *Procedia Comput. Sci.* **151**, 935–940 (2019).
24. Rivoirard, L., Wahl, M., Sondi, P., Berbineau, M. & Gruyer, D. Chain-Branch-Leaf: A clustering scheme for vehicular networks using only V2V communications. *Ad Hoc Netw.* **68**, 70–84 (2018).
25. Zowghi, D. & Gervasi, V. Erratum to ? On the interplay between consistency, completeness, and correctness in requirements evolution?. *Inf. Softw. Technol.* **46**, 763–779 (2004).
26. Jlassi, S., Mammar, A., Abbassi, I. & Graiet, M. Towards correct cloud resource allocation in foss applications. *Futur. Gener. Comput. Syst.* **91**, 392–406. https://doi.org/10.1016/j.future.2018.08.030 (2019).
27. Rivoirard, L., Wahl, M., Sondi, P., Berbineau, M. & Gruyer, D. CBL: A clustering scheme for VANETs. In *VEHICULAR 2017-The Sixth International Conference on Advances in Vehicular Systems, Technologies and Applications*, 19–25 (2017).
28. Leuschel, M. & B, M. ProB: A model checker for B. In *FME 2003: Formal Methods. LNCS 2805* (eds Araki, K. *et al.*) 855–874 (Springer, 2003).

Acknowledgements

The authors acknowledge the support of the CPER ELSAT2020 project, which is co-financed by the European Union with the European Regional Development Fund, the French state and the Hauts de France Region Council.

Author contributions

P.S. and E.C. conceived the protocol formal model. I.A. and P.S. formalized the rules and properties in the model. E.R. and M.G. verified the model, the experiments and the results. I.A. and P.S. analyzed the results. All the authors reviewed the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to P.S.